Ministry of Education of Moldova

Technical University of Moldova

Faculty "Computers, Informatics and Microelectronics"

Report

Laboratory work nr. 3
on Embedded systems

Performed by:
st. gr. FAF-141 D. Lupei

Verified by:
assoc. prof., A. Bragarenco

Chisinau -2016

Topic

ADC conversion. Connecting a temperature sensor

Objectives

- Studying and understanding the AVR microcontroller architecture
- Understanding the principles of environment reading of a sensor
- Understanding the principle of ADC conversion
- Understanding LCD working principles

Tasks

Write a program for the Atmega32 microcontroller in which to get data from the ADC and display in to a terminal connected to the microcontroller.

Short theory

A **sensor** is a device whose purpose is to detect events or changes in its environment, and then provide a corresponding output.
Data transfer from the sensor to the CPU can be either CPU-initiated (*polling*) or sensor-initiated (via *interrupt*). In case it is CPU-initiated, the CPU has to keep checking whether the sensor is ready by reading a status line in a loop. This is much more time consuming than the alternative of a sensor-initiated data transfer, which requires the availability of an interrupt line. The sensor signals via an interrupt that data is ready, and the CPU can react immediately to this request.

Sensor Output	Sample Application
Binary signal (0 or 1)	Tactile sensor
Analog signal (e.g. 0..5V)	Inclinometer
Timing signal (e.g. PWM)	Gyroscope
Serial link (RS232 or USB)	GPS module
Parallel link	Digital camera

Table 1: Sensor output

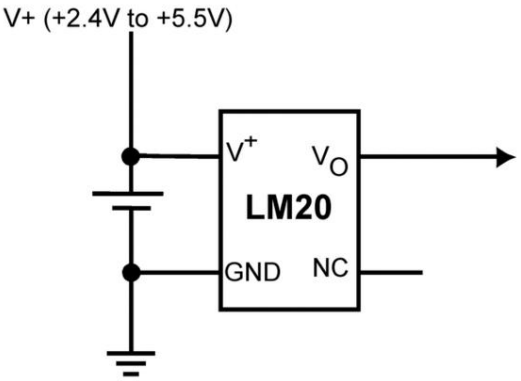
There is the following sensors classification:

- from a robot's point of view, it is more important to distinguish:
    - **Local** or **on-board** sensors *(mounted on the robot)*
    - **Global** sensors *(mounted outside the robot in its environment and transmitting sensor data back to the robot)*
- for mobile robot systems it is also important to distinguish:

    - **Internal** sensors (monitoring the robot's internal state)
    - **External** sensors (monitoring the robot's environment)
- from the point of view of sensor's activity:
    - **Passive** sensors (monitor the environment without disturbing it, for example digital camera, gyroscope)
    - **Active** sensors (stimulate the environment for their measurement, for example sonar sensor, laser scanner, infrared sensor)

LM20 Temperature Sensor

In this laboratory work I use a LM20 temperature sensor for reading environment temperature. LM20 is a precision analog output CMOS integrated-circuit temperature sensor that operates over -55°C to 130°C. The power supply operating range is 2.4V to 5.5V. The transfer function of LM20 is predominately linear, yet has a slight predictable parabolic curvature. The accuracy of the LM20 when specified to a parabolic transfer function is ±1.5 °C at ambient temperature of 30°C.



Scheme 1: *Simplified schematic from datasheet*

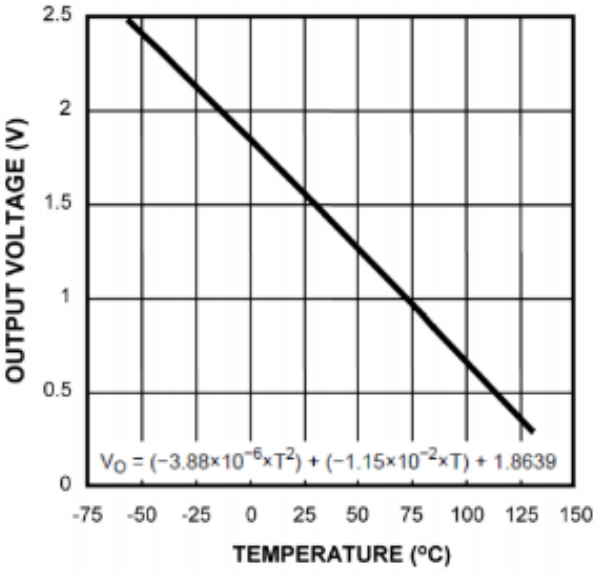


Chart 1: *Output voltage vs. Temperature*

ADC

An **A/D converter** translates an analog signal into a digital value. The characteristics of an A/D converter include:

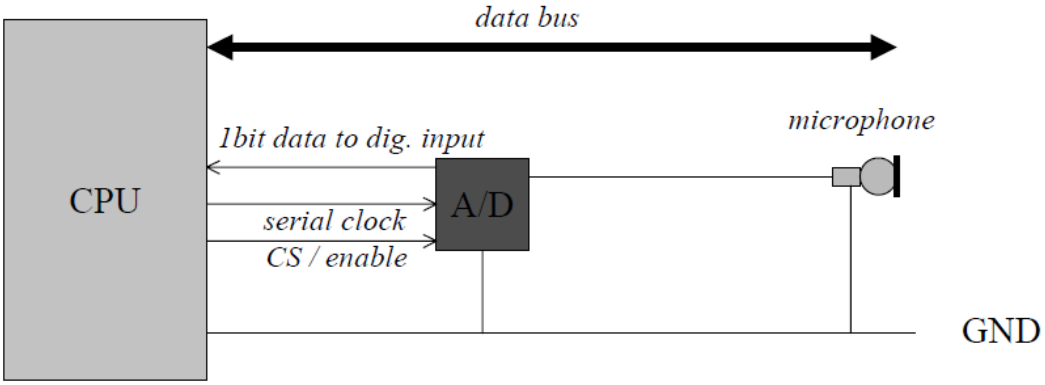
• *Accuracy* – expressed in the number of digits it produces per value (for example 10bit A/D converter)

• *Speed* – expressed in maximum conversions per second (for example 500 conversions per second)

• *Measurement range* – expressed in volts (for example 0..5V)

A/D converters come in many variations. The output format also varies. Typical are either a parallel interface (for example up to 8 bits of accuracy) or a synchronous serial interface. The

latter has the advantage that it does not impose any limitations on the number of bits per measurement, for example 10 or 12bits of accuracy.



Scheme 2: *ADC interfacing*

Most modern MCU including AVRs has an ADC on chip. On AVR MCU the ADC is multiplexed with PORTA that means the ADC channels are shared with PORTA. The ADC can be operated in single conversion and free running mode. In single conversion mode the ADC does the conversion and then stop. While in free it is continuously converting. It does a conversion and then start next conversion immediately after that.
An important part of ADC is the Prescaler, Channels and Registers described bellow.

ADC Prescaler

The ADC needs a clock pulse to do its conversion. This clock generated by system clock by dividing it to get smaller frequency. The ADC requires a frequency between 50KHz to 200KHz. At higher frequency the conversion is fast while a lower frequency the conversion is more accurate. As the system frequency can be set to any value by the user (using internal or externals oscillators). **So the Prescaler is provided to produces acceptable frequency for ADC from any system clock frequency. System** clock can be divided by 2, 4, 16, 32, 64, 128 by setting the Prescaler.

ADC Channels

The ADC in ATmega32 has 8 channels that means you can take samples from eight different terminal. You can connect up to 8 different sensors and get their values separately.

ADC Registers

The registers related to any particular peripheral module (like ADC, Timer, USART etc.) provides the communication link between the CPU and that peripheral. You configure the ADC according to need using these registers and you also get the conversion result also using appropriate registers. The ADC has only four registers.

1. **ADC Multiplexer Selection Register (ADMUX)** – for selecting the reference voltage and the input channel
2. **ADC Control and Status Register A (ADCSRA)** – as the name says it has the status of ADC and is also used for controlling it

3. **ADC Data Register (ADCL and ADCH)** – the final result of conversion is here

Work Process

1. Environment setup by creating the following files: main.c, LM20.c, ADC.c, LCD.c, LM20.h, ADC.h, LCD.h

2. ADC configuration:
   2.1 Initialization:

```
void ADC_init()
{
    ADMUX = (1 << REFS0); // selecting the reference voltage
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // enable ADC with Prescaler=Fcpu/128
}
```

This function selects the reference voltage for ADC, storing it in ADMUX register.
The second instruction enables the ADC (ADEN = 1) and sets the Prescaler.
ADPS2-ADPS0 – these selects the Prescaler for ADC
Here is a visual representation of the ADCSRA (ADC Conrol and Status Register A):

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

2.2 Conversion function:

```
int ADC_GetData(char channel)
{
    while(ADCSRA & 1 << ADSC); //wait until ADC is busy
    channel &= 0b00000111; //normalize the channel, leave last 3 bits LSB
    ADMUX = (ADMUX & ~(0b00000111)) | channel; //apply the channel to the ADMUX with protection of
configuration bits
    ADCSRA |= (1<<ADSC); //start conversion
    while (ADCSRA & (1<<ADSC)); //wait until conversion is complete
    return (ADC);
}
```

This function takes as parameter the channel for data reading (analog signal from the temperature sensor) which is 7 in our case. It waits until the ADC is not busy, then normalize the channel, leaving the last 3 last significant bits. After that, the channel is applied to the ADMUX with protection of configuration bits, and the conversion starts. In the last *while* is waited until the conversion is completed, and the obtained ADC value is returned.

3. LM20 configuration: incorporate the ADC_init() function into the LM20_init(), and the ADC_GetData(7) in the LM20_GetTemp() function. The argument 7 means the channel the ADC will read data from.
4. LCD configuration

   The LCD drivers have been taken from the following source: http://tinyurl.com/peterfleury The libraries contains basic routines for interfacing a HD44780U-based text LCD display, such as: *lcd_init*, *lcd_putc*, *lcd_delay* etc.

5. Write main function which combines the written driver-functions and consists of the initialization part of the ADC and LCD, and the infinite loop where the temperature value is obtained, converted and displayed on the LCD screen.

   Here how the temperature is obtained and converted:

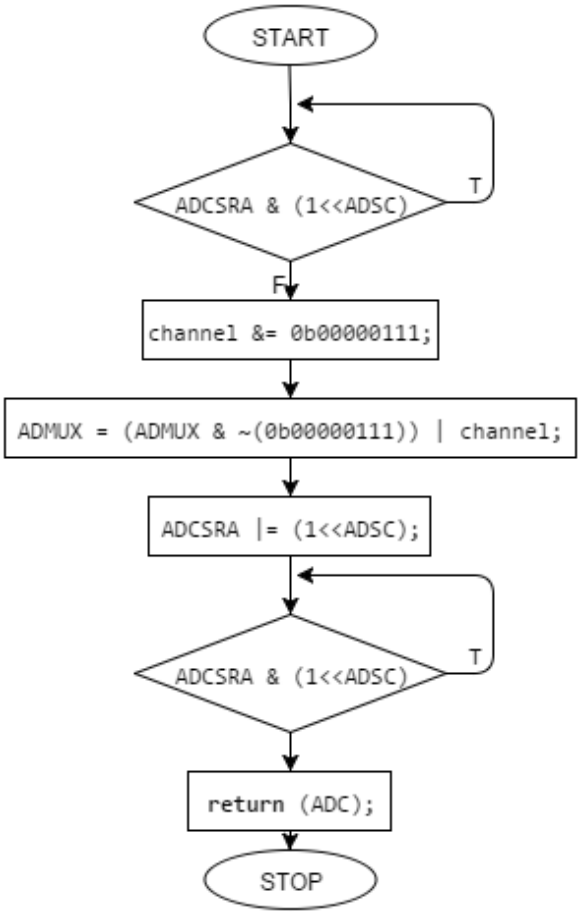
```
ADCvalue = LM20_GetTemp();
tempCelsius = (param - ADCvalue) / param0; // param0 = 2.4; param = 382;
```

There is also calculated the equivalent temperature in Fahrenheit units by the formula:

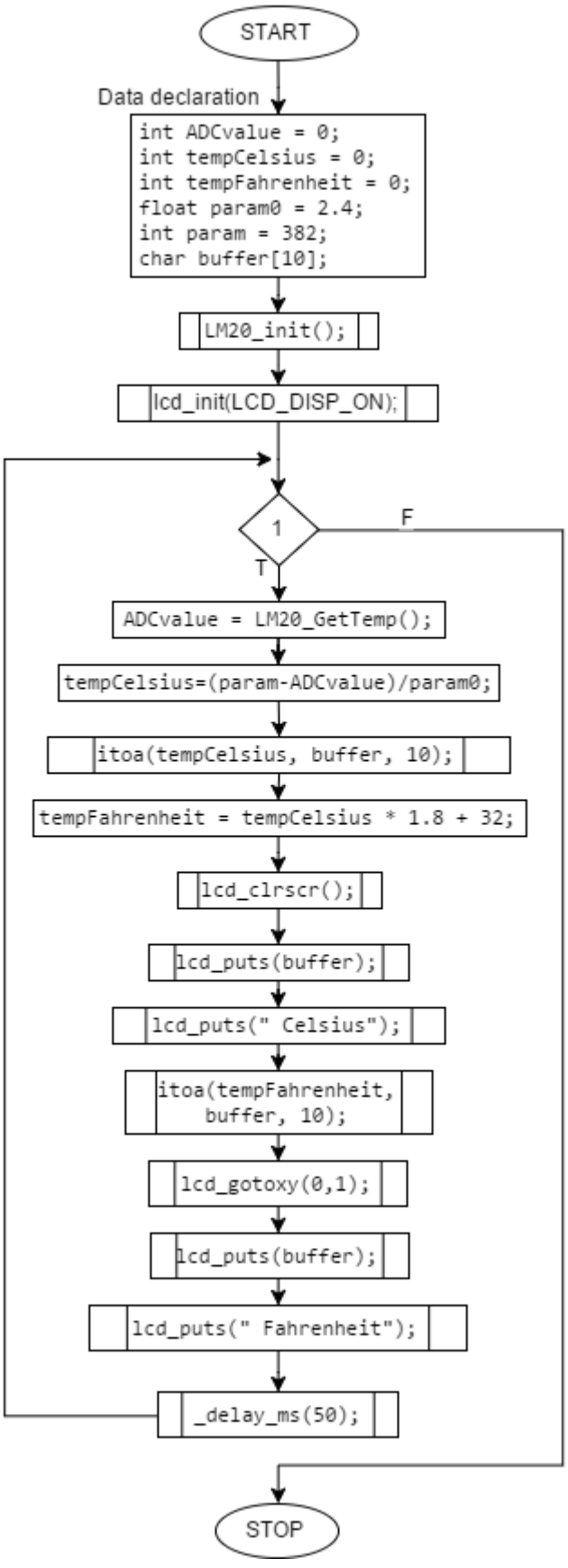
$$T_{(°F)} = T_{(°C)} \times 1.8 + 32$$

6. Building the solution and importing the .elf resulting file in Proteus

Flow charts

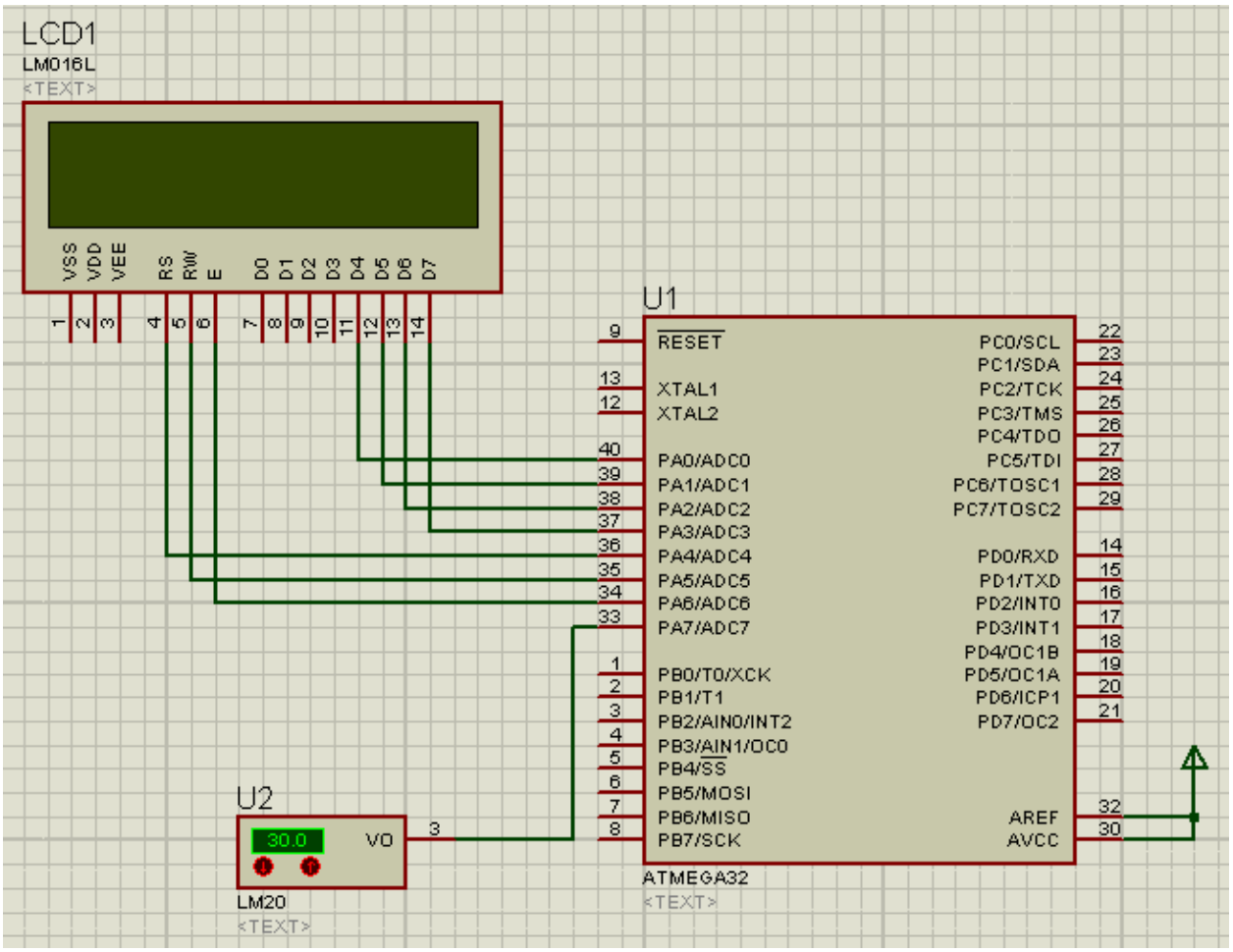


Flowchart 1: *function ADC_GetData(channel)*

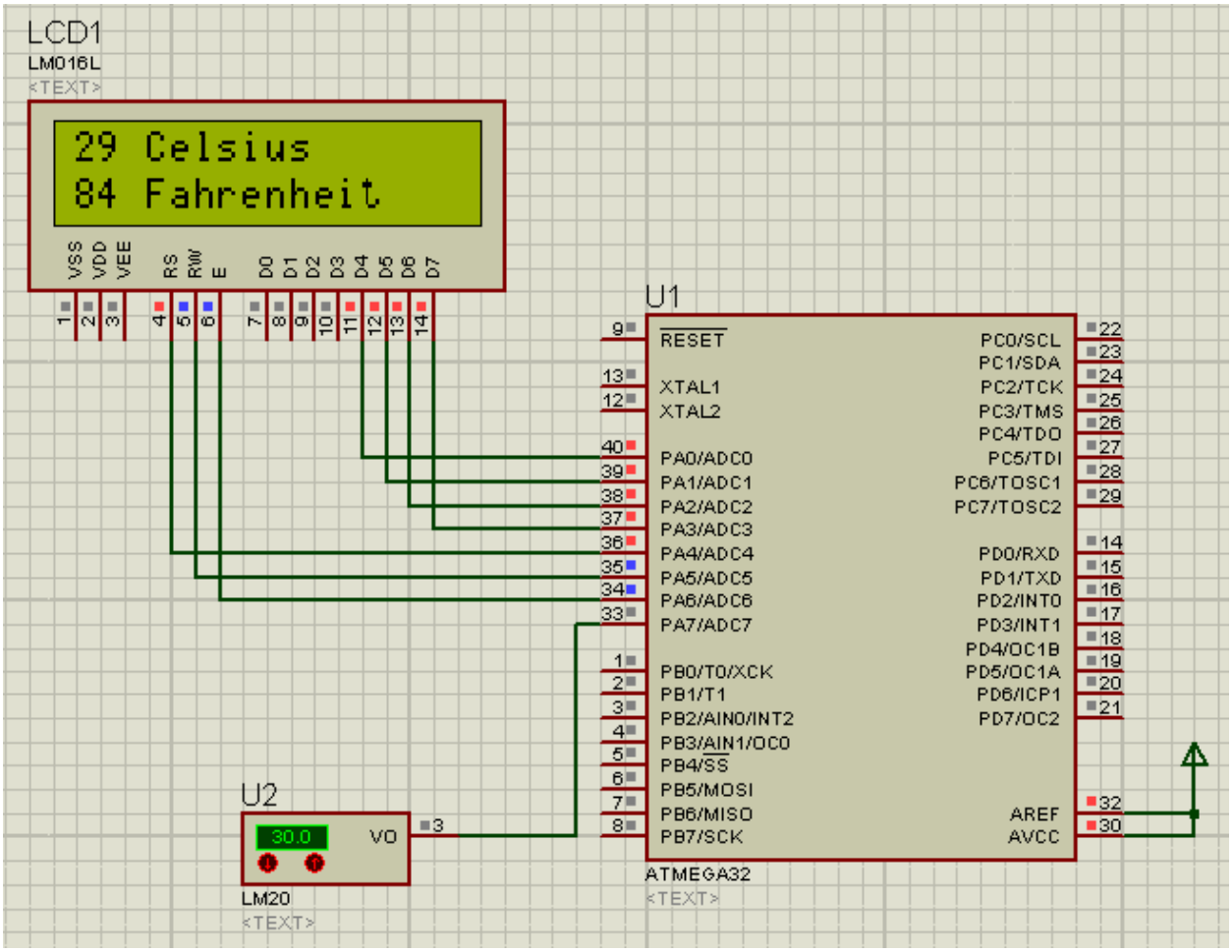


Flowchart 2: *function main( )*

Circuit Scheme



Simulation:



Conclusion:

In this laboratory work has been learnt the principles of how a sensor collect environment information in forma of a analog signal and how it is further converted in a digital value with the help of the ADC. ATmega32 has an incorporated ADC which is multiplexed with PORTA that means the ADC channels are shared with PORTA. It plays a great role in environment inspection process.

Bibliography

- M.A. Mazidi, "*AVR microcontroller and embedded system using C and assembly language*", 2009
- Thomas Bräunl, "*Embedded Robotics*"
- http://www.embedds.com/sensing-temperature-using-avr/
- http://www.microlab.club/search/label/Laborator

Appendix

ADC.h

```
#ifndef ADC_H_
#define ADC_H_

	void ADC_init();
	int ADC_GetData(char channel);

#endif /* ADC_H_ */
```

LM20.h

```
#ifndef ADC_H_
#define ADC_H_

	void ADC_init();
	int ADC_GetData(char channel);

#endif /* ADC_H_ */
```

ADC.c

```
#include <avr/io.h>

void ADC_init()
{
	ADMUX = (1 << REFS0); // selecting the reference voltage
	ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // enable ADC with Prescaler=Fcpu/128
}

int ADC_GetData(char channel)
{
	while(ADCSRA & 1 << ADSC); //wait until ADC is busy
	channel &= 0b00000111; //normalize the channel, leave last 3 bits LSB
	ADMUX = (ADMUX & ~(0b00000111)) | channel; //apply the channel to the ADMUX with protection of
configuration bits
	ADCSRA |= (1<<ADSC); //start conversion
	while (ADCSRA & (1<<ADSC)); //wait until conversion is complete
	return (ADC);
}
```

LM20c

```
#include "ADC.h"
#include <avr/io.h>

void LM20_init()
{
	ADC_init();
}

int LM20_GetTemp()
{
	ADC_GetData(7);
}
```

main.c

```
#define F_CPU 8000000UL
#include "LM20.h"
#include "LCD.h"
#include <util/delay.h>

int main(void)
{
	 //char ch;
	int ADCvalue = 0;
	int tempCelsius = 0;
	int tempFahrenheit = 0;
	float param0 = 2.4;
	int param = 382;
	char buffer[10];

	LM20_init();
	lcd_init(LCD_DISP_ON);


while(1)
	{
		ADCvalue = LM20_GetTemp();
		tempCelsius = (param - ADCvalue) / param0;
		itoa(tempCelsius, buffer, 10); // converts tempCelsius numeric value into a string and stores it in buffer,
base 10
		tempFahrenheit = tempCelsius * 1.8 + 32; // formula of Celsius - Fahrenheit conversion
		lcd_clrscr();
		lcd_puts(buffer);
		lcd_puts(" Celsius");
		itoa(tempFahrenheit, buffer, 10);
		lcd_gotoxy(0,1); // go to new line
		lcd_puts(buffer);
		lcd_puts(" Fahrenheit");
		_delay_ms(50);
	}
}
```